An Open Framework for Signal Routing, Processing, Logging and Visualization

In Automotive, Energy and Industrial applications

Guido Del Vescovo[1], Maurizio Paschero[2], Antonello Rizzi[3], Fabio Massimo Frattale Mascioli[4]

Information Engineering, Electronics and Telecommunications Department, University of Rome "La Sapienza"
Polo per la Mobilità Sostenibile Laboratories, Via Eudossiana 18, 00184, Rome, Italy

{[1]guido.delvescovo; [2]maurizio.paschero; [3]antonello.rizzi; [4]fabio.mascioli}@pomos.it

Abstract

The RAMSES (Reusable Automotive Signal Elaboration System) project is an ensemble of software libraries and applications aimed at gathering signal time series coming from an heterogeneous set of sensors and devices, with the capability of built-in preprocessing, logging, transmission and visualization of the collected data. The main purpose is to enable the fast development of systems for applications such as control, monitoring, diagnostics, power management and actuation, especially in automotive, industrial and smart grid applications.

Keywords

Signal Routing; Software; Automotive

Introduction

The digital processing of data in automotive, industrial, and smart grid applications is ubiquitous nowadays [Puschnig (2004)], [Rodelgo-Lacruz (2007)], [Katramados (2008)]. A great number of devices like combustion engines, electric motors, battery packs and power inverters are provided with embedded control units communicating with the external world through some digital link. Measures collected by sensors are of prime importance for the safe and proper operation of power systems. Prototyping and development of such systems requires the possibility of monitoring and keeping a great number of quantities under control.

The market offers a highly differentiated variety of solutions for measurement and control applications. Some companies offer solutions of closed or semi-closed type, based on the idea of developing the whole system by interconnetting and programming units of the same brand, or limited number of brands, with proprietary developing environments. Often, visual programming languages are provided. This kind of solution enables rapid development and deploying cycles, especially if developers trained on the proprietary programming environment are available. On the other hand, the possibility to choose of the devices to be employed is somewath restricted, since the cost of hardware and software licenses is often high and only a limited control is possible over the actual code running on the targets, as well as over the communication protocols and operating systems and graphic environments to use.

The opposite approach in chosing the building blocks of the system is to employ open solutions based on standard programming languages like C/C++ or Java, and operating systems distributed under free software licenses, like Linux. This approach offers a vast choice in the determination of both the hardware and software platform to be adopted. In particular, embedded devices capable of running a Linux operating system and programmable in C/C++ language are widespread, for example in the form of SBC (Single Board Computer) with a selection of peripherals for communicating with the outside world. Often it is possible to stack additional boards over the main SBC, enabling the user to add the desired features. Due to the large number of manufacturers providing such devices, and their variety in cost and processing power, it is often possibile to build a system which is actually tailored for the target application, with very optimized code. The main disadvantage is the cost of the software development, especially if developers have to start from low level routines.

The proposed systems aims at providing a general framework to speed up the development of prototypal or deployed applications where the platform is constituted by embedded or general purpose computers running the Linux operating system. The GCC toolchain is mainly employed to build the code, which is almost exclusively written in C language, with limited employ of C++ for some graphic tools.

Overview of the RAMSES System

RAMSES is mainly conceived as a framework to put a signal routing and processing software system together, by building a certain number of executables called *modules*. The different modules run concurrently as separated processes interacting through some interprocess communication (IPC) mechaninsms. Each module is usually in charge of handling a flow of data coming from an input device or going to an output device. These I/O devices can be physical or logical, including ADC/DAC channels and digital data links such as serial ports or CAN/LAN interfaces, for instance. Other devices could be represented by files on disk or graphical windows appearing on some display. Modules in charge of performing processing operations on data circulating inside the system are also possible.

The configuration of modules is almost totally prescribed through XML files stored on disk. A module may read one or more configuration file, if needed, at startup. Only a minimal set of parameters is passed through the command line, as explained in the following paragraph. Once initialized, a module usually enters an infinite loop in which it reads and/or writes data from/to the centralized data handling facilities. The main IPC mechaninsm enabling data exchange between modules is a shared memory segment, to which access is regulated by means of semaphores. Another IPC facility is provided in the form of message queues enabling modules to transmit and receive start/stop commands, status information or other asynchronous communication. At the current state of implementation, message queues are available but not yet used by modules which actually interact only through the shared memory.

A key concept in the understanding and practical use of the system is the RAMSES *session*. The concept of session is actually twofold: the session can be seen as the ensemble of RAMSES modules running together during the operation. On the other hand, the *session folder* is also the directory containing the configuration files required by the just mentioned modules. The session folder is thus the static counterpart of a running session, almost defining the complete behaviour of the system during operation. More than one RAMSES session can be run at once. Different sessions are actually identified by integer numbers. By naming convention, a folder session name must begin with the indication of a specific integer number. All session folders are placed in a conventional directory, namely */usr/share/ramses* in the current implementation. As an example, a valid session folder full path would be: */usr/share/ramses/01_my_session*. Each RAMSES module must receive the session folder name (not the full path) as its first and mandatory command line argument. The only other possibile command line argument, optional when expected, is a value expressing the operating rate of the module in milliseconds. The structure of a command line call used to start a RAMSES module is thus:

```
module_name session_folder [rate]
```

Example:

```
my_module 01_my_session 20
```

The RAMSES project is conceived as an extensible system, in the sense that it provides the IPC facilities and a standardization of module configuration and launching style, as well as some readymade modules which implement different functionalities, from basic session management (*core* modules) to implementation of I/O operations with hardware from specific manufacturers. The existing modules serve for both practical use and providing a set of development examples to facilitate the users to create their custom modules.

At this time, the RAMSES project includes the following list of implemented libraries and modules:

- The *libramses* library. This library can be regarded as the core of the system, as it implements the IPC mechanisms and the means to access the shared data in a safe way, hiding the operating system details. It also contains the base functionalities to dump data to file descriptors. Creation, disposal of a RAMSES session, connection to an existing session, reading, writing and dumping of data, message exchange are done by the user through elegant and safe API (Application Programming Interface) calls.
- The *libxmlutils* and *librsescoreconf* libraries, providing some middle level support for the task of reading XML configuration files. These libraries, in turn, rely on the well known and extremely portable *libxml2* for low level support.
- The *libsigmap* library, providing the base support to create modules capable to exchange data through I/O devices, with the capability to apply user defined piecewise linear mapping functions on the values.

- The *core* modules. A set of readymade modules provides basic functionalities to create, destroy, debug, join, run sessions with particular modes of management of the shared memory.
- Graphic modules and libraries. Some readymade graphical tools are available. The *librsnxgraph* library implements some typical widgets for monitoring purpose, like gauges, led arrays, 7-digit displays. These widgets are built on top of the *nano-X* server API. A general purpose monitoring panel built on top of the *wxWidgets* framework is also available.
- Hardware I/O modules. Some modules implementing I/O functionality for some specific hardware or protocols are available, namely ADC/DAC and digital input with Diamond Systems, Inc. devices, and CAN support for devices implementing the SocketCAN standard. The list is growing.

One of the strengths of the RAMSES system resides in the usability of data. Once a module for reading data from a specific piece of hardware has been written, data is immediately accessible in a number of ways, like value printing to console, writing of disk logs, graphic visualization. In fact, RAMSES is conceived as a platform for providing ease of access to data gathered from different sources, for elaboration with DSP tools or for feeding Machine Learning systems, along with monitoring and inspection purposes.

The Shared Memory Layout

The main interaction mechanism between modules is the controlled read/write access to the shared memory segment. Each RAMSES session is associated with a unique shared memory segment, accessible to each module. In the RAMSES jargon, a *signal* is a monodimensional time series. The shared memory holds some static information, like total number of signals, name, unit, range. This information never changes during the execution of the session. The dynamic part of the shared memory actually contains the sampled values, representing the time series associated to each signal.

A *slot* of the shared memory is an ordered set of values, actually a vector whose dimension equals the number of signals, accompanied by a timestamp. In other words, a slot contains the last known value for each signal, at the instant indicated by the timestamp. The shared memory contains a user defined number of slots, managed as a ring buffer. Usually, slots are equally spaced in time, yielding a set of time series presented under a unified sampling clock. This enhances the usability of data by DSP or Machine Learning algorithms requiring the set of time series to be aligned to a unique time grid. Another useful way of viewing the shared memory content is as a sequence of uniformly sampled determinations of a *status vector* representing the current values of the signals of interest. It is worth remarking that, in case of signals coming from sources producing data at different rates, the sources with slower rate are treated in a sample and hold fashion. That is, the last known values are carried through the next slot when the global sampling clock occurs. This introduces a certain amount of redundancy, which yields the enhanced ease of access and elaboration of data.

The ring buffer holding signal values can also be regarded as a matrix, the rows represent the slots progressing with time, the columns represent the signals, whose number and order is fixed throughout the session. The most usual way to orgainize a RAMSES session is running a certain number of modules cooperated under a producer/consumer paradigm. Some modules acquire data from devices or user input, writing signal values in specific columns of the shared memory, other modules read values from the shared memory and route them towards output devices. Both producers and consumers could perform some processing of data before writing them to the shared memory (producers) or after reading them from it (consumers). Modules which are meant for internal processing can be also implemented. A module of this type may read values from some columns of the shared memory, process them, and write the results to other columns. Such a module actually is both a consumer and a producer. Figure 1 shows a session featuring two producers, two consumers, and one module which is both a consumer and a producer.

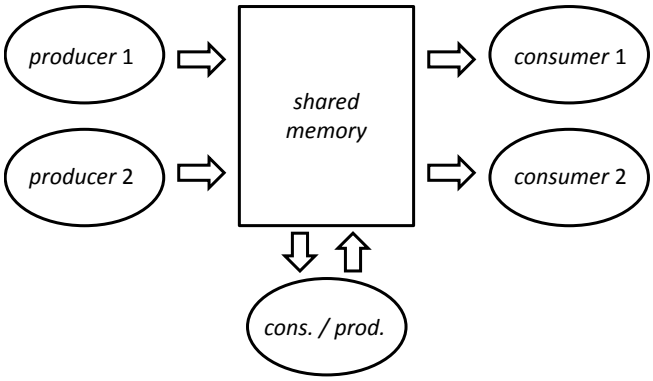

FIG. 1 RAMSES SESSION WITH FIVE MODULES

In order to describe the complete layout of the shared memory segment, some values representing the length in bytes of some fields or aggregate of fields have been defined. The given lengths are valid for the current implementation on most 32-bit systems. Let *S* be the current number of signals and *B* the size of the ring buffer, both of which are user defined at configuration time. The size of a slot (or row) is *R*=8+4**S*. The first 8 bytes hold the timestamp, while 4**S* bytes are required for the storage of *S* 32-bit floating point values. The size of the static information associated to a single signal is *I*=32*2+4*2=72. Two strings of 32 bytes store signal name and unit, while 2 floats store the expected minimum and maximum values at runtime. If some information is unavailable, the relative fields can be left blank.

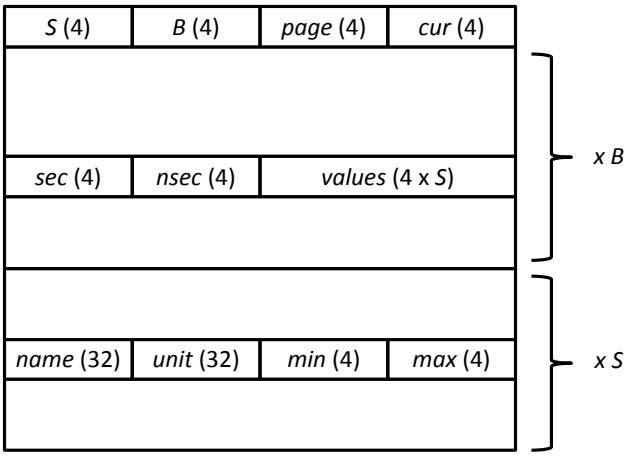

FIG. 2 SHARED MEMORY LAYOUT

The size of the whole shared memory segment is *M*=16+*R***B*+*I***S*. The first 16 bytes contain important information, namely the *S* and *B* values (internally needed by *libramses* functions to compute offsets), the current position of the cursor scanning the ring buffer and the number of times ring buffer has been filled and cursor reset to start (number of *pages* written). The number of pages is used by *libramses* dumping function to detect possibile lost signal slots and inform the user. The heading 16 bytes are followed by the dynamic ring buffer (size is *R***B*) and by the static signal information part (size is *I***S*). As an example, a session handling 20 signals, with a ring buffer size of 64, would lead to a total shared memory size of approximately 6.92 kB. Figure 2 shows the shared memory layout, with byte size of each field reported in parentheses.

The Core Modules

Some readymade modules are of prime importance in the RAMSES system. The following overview will present these modules and explain their purpose.

Session management

The modules implementing session management are distingued by the **ses* suffix in their name, namely, *createses*, *infoses*, *runses*, *destroyses*, *serses*. The *createses* module is in charge of interacting with the operating system in order to allocate the IPC mechanisms required for the session, in particular the shared memory segment. After its operation, the module ends its execution and returns. Every configuration information for session creation is contained in the file *createses.xml*, located inside the session folder. For example, launching the following command:

```
createses 01_my_session
```

produces the following actions: the module opens the configuration file inside the indicated folder and parses it. If the file is correct, the module is able to read the number of signals (*S*), the ring buffer size (*B*), name, unit and range information associated to each signal. After successful completion of configuration file parsing, the shared memory segment and the message queue are created. These IPC mechanisms are internally associated with the identifier 1 (prefix in name of the folder), and other modules are able to find them thanks to the functions made available by the *libramses* API.

The *infoses* module, mostly available for debugging and inspection purposes, simply dumps to standard output the static information associated to signals. The *runses* module, as suggested by its name, is in charge of actually *running* the session by performing the increment of the ring buffer cursor at a fixed rate. The rate at which runses is launched is the superior limit to the sampling rate of signals. A typical launch command would be:

```
runses 01_my_session 20 &
```

If the session has been previously created with *createses*, this command would cause *runses* to run in background, advancing the cursor of the ring buffer every 20 ms. Once this mechanism is active, other modules can write and read signals, implementing the producer/consumer operation. The runses module does not need any configuration file.

The *destroyses* module is in charge of deallocaing the IPC mechanisms created by *createses*, and also serves as a global stop, because the deallocation of the shared memory is detected by a well written RAMSES

module in a short time, and this module should respond with a correct termination of execution.

Serial Connection

The *serses* module can be used in substitution of the *runses* module when a serial link is established between two different computing platforms running RAMSES. More precisely, it is possible to connect the two systems in a leader/follower fashion. The first system, the leader, normally runs the session with *runses*, and sends the entire content of the signal slots through a serial port, using the *serdump* module. The system at the other end of the serial link, the follower, runs *serses* instead of *runses*. By this way, the follower fills its shared memory segment with an exact copy of the signal values received from the leader through the serial port. This mode of operation is particularly useful in the development stage of a prototype, when it is convenient to do expensive inspection of data (complete logs on disk or graphic visualization, for instance) on a PC platform (the follower), in the case that it is unfeasible or unconvenient to do it on the embedded platform under test (the leader).

Altough the readymade modules presented in this section are able to fit most needs, nothing prevents the user from developing custom session management modules making use of the *libramses* API.

Disk Logging

A powerful readymade RAMSES module, called *diskdump*, is capable to dump the whole time series of signals flowing through the shared memory, accompanied by the timestamps, on user friendly log files stored on persistent media. Many options for formatting the output are available to the user at configuration time. For increased readability, the timestamps are reported as relative values, with respect to a time origin associated with the single file. The time origin itself is saved too, so that absolute times can be easily reconstructed. The size of each log file can be limited for ease of transfer and opening. When a file is closed, acquisition continues on the next file.

Hardware Support

Hardware support plays a key role in the development of the RAMSES system. At this time, support modules for a few platforms have been written and tested but, due to the ease of development guaranteed by the API structure, new support modules can be rapidly added. The performance of the project largely depends on success in supporting a high number of devices from well known manufacturers. Both x86 and ARM targets have been successfully used so far to run RAMSES sessions. In the following, a list of experimented hardware targets is reported.

FIG. 3 VIRTUAL DASHBOARD WITH GPX21

GPX21, SBC with TFT Display

The GPX21 manufactured by the Italian company Engicam s.r.l. is a single board computer equipped with a Freescale ARM CPU and an additional touchscreen TFT display. Many I/O peripherals are available on the board, including LAN, CAN, RS-232, GPS, GSM. The GPX21 has been extensively used to provide sports cars prototypes [Di Giacomo (2010)], also used in student competitions, with wirtual dashboards. This context made possibile the development of the early versions of the RAMSES system. Predecessors of the actual modules were created, along with many I/O modules including experiments of GPS data collection and gesture recognition from touchscreen, along with an early CAN receiver module and with a graphic module displaying the dashboard through the *nano-X* server [Del Vescovo (2012)]. Many of these old modules need to be revamped in order to work with the current RAMSES architecture. Figure 3 shows a virtual dashboard realized with the GPX21.

Helios, PC/104 Platform

The Helios board manufactured by Diamond Systems, Inc. is a PC/104 platform allowing stacking additional boards and enclosing the whole system in a rugged box with a convenient I/O panel. Diamond Systems, Inc. provides a custom Linux distribution and a unified driver to access each functionality of its boards

through a well organized set of API calls in C language. RAMSES modules have been written to deal with ADC, DAC and digital input functionalities of Diamond boards. These modules have been successfully deployed and tested on the Helios board, where a RAMSES session is used to perform the task of power train management in the project of a parallel hybrid offroad vehicle which is currently under development. Figure 4 shows the I/O panel of the enclosure for the Diamond boards.

FIG. 4 I/O PANEL OF DIAMOND PC/104 SYSTEMS

Beaglebone, Open Hardware Linux Box

The Beaglebone, part of the Beagleboard project, is an open hardware project featuring an ARM processor manufactured by Texas Instruments, Inc. The functionality of the Beaglebone can be expanded by the addition of daughterboards called *capes*. The platform which is currently the main target of RAMSES development has a CAN cape to communicate through the CAN bus. Since the Linux operating system provided with the board (Angstrom distribution) implements the modern SocketCAN interfacing support, a RAMSES module capable to encode/decode CAN messages through this protocol has been developed. This module is theoretically capable to work on any platform providing access to the CAN functionalities through the SocketCAN network interface. This system has been developed as part of a project consisting in a battery pack recharge system exploiting a combustion engine as the originary power source, for vehicles of series hybrid type.

API Example

The development of new RAMSES modules is made easy by the high level API provided by *libramses*. At its simplest, a C program that joins a session and reads a value can be written as follows:

```
#include <stdio.h>
#include <ramses.h>
int main() {
int res;
float val;
res = rses_join_session(1);
if (res != RSES_SUCCESS) {
    printf(“Unable to join session\n”);
    return -1;
    }
res = rses_get_signal_value(3, &val);
if (res != RSES_SUCCESS) {
    printf(“Unable to read value\n”);
    return -1;
    }
printf(“Value = %f\n”, val);
return 0;
}
```

This program joins the session having id number equal to 1, and reads a value from the fourth column of the shared memory (index base is always 0). Of course, if the session does not exists or if the number of signals is less than 4, the relative function will fail. In a real world program, the session indentifier should be extracted from the session folder name passed as a mandatory argument, and the number of available signals should be known by calling the appropriate function.

Considerations about Operating Rates

The RAMSES software systems is conceived and designed to run on platforms providing a Linux operating systems and a certain number of facilities (displays, I/O peripherals) able to grant a high level of accessibility of data for use and inspection. In order to be generally portable to the largest possible number of devices, RAMSES does not rely by default on realtime capabilities of the operating system. Thus, it is not to be compared with custom hardware running a single

program or a reduced set of scheduled tasks for high processing rate, like the case of combustion engine control units. Rather, the RAMSES system is meant to convey data at middle rates (in the order of tens of milliseconds) for purposes like monitoring, measure collection, power management in vehicles, smart grids and domotics.

This does not mean that the RAMSES modules can not be able to handle data at high rates. For example, the ADC support for the Diamond boards internally exploits the interrupt based sampling provided by the platform driver to receive and process the analog samples at high rates. However, these samples are processed by DSP blocks like moving average, lowpass or envelope filters, FFT methods, or by frequency metering algorithms. Then, the aggregated results of this processing are conveyed to the RAMSES shared memory at lower rates.

Despite of the middle rate nature of the RAMSES system, the addition of realtime capabilities to the system is planned. When this task will be completed, RAMSES will be able to exploit the capability of realtime Linux kernels such as RTAI, in order to increase the operating rates by one or two orders of magnitude.

Future Work

The RAMSES project is still at an early stage of development. Features that are planned or under development include:

- Realtime capabilities to run accurately at higher rates when a realtime kernel is available.
- A built-in facility for applying DSP algorithms to the handled time series at running time. Desired algorithms include spectral analysis through FFT, extraction of envelope or RMS value from sinusoidal waveforms, frequency metering, denoising. These DSP techniques provide a further signal conditioning stage, able to yield an even more usable representation of the monitored quantities both at running time and offline.
- A built-in facility for applying Machine Learning techniques to the handled time series at running time. The use of Machine Learning techniques, such as neurofuzzy systems and SVM, employed to solve clustering, classification, function approximation and forecasting problems, would enable the possibility to perform in real time intelligent tasks like diagnostics of undesired conditions, optimization of power management, actuation of specific actions in response to some recognized pattern.
- Extension of readymade hardware support, in order to enable the fast development of solutions on hardware devices from a reasonably large set of manufacturers.
- Publication of the software repository in order to make it available for everyone under the GPL license.

Conclusions

The RAMSES project is a software framework that can effectively speed up the process of developing a complete solution when targeting hardware running a Linux operating system. The RAMSES modules take care of giving transparent access to the collected data, by solving the issues derived from the gathering of time series coming from different sources operating at different rates. Built-in output modules like disk logger and screen visualizers give an immediate inspection on data collected. RAMSES is also a perfect tool to produce data collections obtained from measurement campaigns, with the aim of investigating the application of DSP or Machine Learning algorithms on them.

ACKNOWLEDGMENT

This work has been funded by the Pole for Sustainable Mobility (POMOS) of Regione Lazio. POMOS is an initiative taken by the Regional Administration of Lazio to promote the start-up of an industrial district specialized on Sustainable Mobility. POMOS acts a connection point for the interaction of different subjects (Academia, Industries, Regione Lazio, Municipalities, PA), which constitute a network of subjects interested in the sector of Sustainable Mobility.

REFERENCES

Del Vescovo, G., Paschero, M., Rizzi, A., Mascioli, F.M.F. "An open software system for signal routing and processing in hybrid vehicles." in Industrial Electronics (ISIE), 2012 IEEE International Symposium on, 2012 , Page(s): 1702-1707.

Di Giacomo, V., Martellucci, L., Mascioli, F. M. F., and Sgreccia, S. "Bizzarrini p538 eco targa project desing and

construction of a parallel hybrid." in Proceeding of XIX International Conference on Electrical Machines - ICEM 2010, Rome, 2010.

Katramados, I., Barlow, A., Selvarajan, K., Shooter, C., Tully, A., and Blythe, P. "Heterogeneous sensor integration for intelligent transport systems." in RTIC 2008 and ITS United kingdom Members' Conference in Road Transport Information and Control, 2008, pp. 1–8.

Puschnig, A., and Kolagari, R. T. "Requirements engineering in the development of innovative automotive embedded software system." in Proceeding of the 12-th International Conference on Requirement Engineering, 2004, pp. 328–333.

Rodelgo-Lacruz, M., Gil-Castineira, F. J., Gonzales-Castinero, F. J., Pousada-Carballo, J. M., Contreras, J., Gomez, M. V., Bueno-Delgado, M., Egea-Lopez, E., and Garcia-Haro, J., "Base technologies for vehicular networking applications: review and case studies." in Proceeding of IEEE International Symposium on Industrial Electronics, ISIE, 2007, pp. 2567–2572.

Guido Del Vescovo is a post doctoral research associate at the Information Electronics and Communications Department (DIET) of the University of Rome "La Sapienza" since 2008. He received the Laurea degree in Electronics Engineering in 2004 and his Ph. D in Information and Communication Engineering in 2008 from the University of Rome "La Sapienza". His major fields of interest include supervised and unsupervised data driven modelling techniques, neural networks, fuzzy systems, evolutionary algorithms and granular computing. He is a developer of the SPARE library (http://libspare.org), an open source C++ project for pattern recognition and machine learning.

Maurizio Paschero is a post doctoral research associate at the Dipartimento di Ingegneria dell'Informazione, Elettronica e Telecomunicazioni (DIET) of the University of Rome "La Sapienza", and since September 2008, he has worked in the Polo per la Mobilità Sostenibile (POMOS) Laboratories. He received the Laurea degree in Electronics Engineering in July 2003 and the Ph.D in Information and Communication Engineering in May 2006 from the University "La Sapienza" of Rome and the Ph.D in Mechanical Engineering in March 2008 from Virginia Polytechnic Insitute and State University. His major fields of interest include circuital modeling of multi-physic systems, intelligent signal processing, control of hybrid powertrain, smart strucuture, stability of structure. He is author or coauthor of more than 10 scientific pubblications on international journals and conferences.

Antonello Rizzi received the Dr. Eng. degree in Electronic Engineering from the University of Rome "La Sapienza" in 1995 and the Ph.D in Information and Communication Engineering in 2000, from the same University. In September 2000 he joned the "Information and Communication" Department (INFO-COM Dpt.) of the University of Rome "La Sapienza" as an Assistant Professor. Since July 2010 he has been with the "Information Engineering, Electronics and Telecommunications" Department, in the same University. His major fields of interest are in the area of Soft Computing, Pattern Recognition and Computational Intelligence, including supervised and unsupervised data driven modelling techniques, neural networks, fuzzy systems and evolutionary algorithms. His research activity concerns the design of automatic modelling systems, with particular emphasis on classification, clustering, function approximation and prediction problems. In particular, he is currently working on classification and clustering systems for structured patterns, graph matching, symbolic inductive modelling systems and Granular Computing. He is author of more than 80 publications on international journals and conferences.

Fabio Massimo Frattale Mascioli was born in Rome, Italy, on June 13, 1963. He received the Laurea degree in Electronic Engineering in 1989 and the Ph.D.degree in Information and Communication Engineering in 1995 from the University "La Sapienza" of Rome. In 1996, he joined the DIET Department (ex INFOCOM) of the University "La Sapienza" of Rome as Assistant Professor (Rearcher). Since 2000, he has been Associate Professor of Circuity Theory at the same department. His research interest mainly regards neural networks and neuro-fazzy systems and their applications to clustering, classification and function approximation problems. Currently, he is also working on circuit modelling for vibration damping, energy conversion systems, electric and hybrid vehicles, energy-mobility integrated systems. He is authir or co-authir of more than eight papaers presented at international conferences or published in international scientific literature. Since 2007, he has been the scientific director of the "Polo per la Mobilità Sostenibile della Regione Lazio" (Sustainable Mobility Polo of Lazio Region).